JxBrowser

Programmer's Guide

Version: 2

TeamDev

© 2000-2010 TeamDev Ltd.

Table of Contents

1. Introduction **1**
2. Requirements **2**
2.1. Windows: 2
2.2. Linux: 2
2.3. Mac OS X: 2
3. Browser Engines **3**
3.1. Mozilla Firefox browser 3
3.2. Internet Explorer browser 3
3.3. Safari browser 3
3.4. Libraries Dependency 3
4. Navigation to URL **6**
5. Navigation to URL with POST data **7**
6. Working with Browser Events **8**
7. Pop-up windows management **9**
8. Working with Browser Dialogs **10**
9. JavaScript execution **11**
10. Cookie management **12**
11. Change user-agent string **13**
12. Using JxBrowser with Java Web Start **14**
13. Using JxBrowser with Java Applet **18**
14. JxBrowser System Properties **19**
15. Support **20**

Chapter 1. Introduction

All Java programmers who have ever developed a Web site testing tool or desktop Java application with an HTML viewer needed a library that could load, display an HTML document and access its content. Of course, the implementation of such library from scratch may take a lot of programming and testing efforts. Therefore, integrating an existing browser is a preferable way.

JxBrowser is a cross-platform library that enables seamless integration of Mozilla Firefox web browser into Java AWT/Swing applications on Windows, Linux, Mac OS X platforms. Additionally it allows integration of Microsoft Internet Explorer on Windows and Apple Safari on Mac OS X platform.

So, using JxBrowser you can define which browser engine your Java application should use on different operating systems. You can easily switch between browser engines and work with them via the same API.

Chapter 2. Requirements

You can see the list of JxBrowser supported platforms at JxBrowser Support Site

2.1. Windows:

1. Microsoft Windows XP/Vista/7 or higher with x86 or x64 arch;
2. Sun Microsystems Inc. Java JDK 1.5+ 32-bit or 64-bit version.

2.2. Linux:

1. Linux Kernel 2.6;
2. GTK 2.14+ with all dependent packages;
3. gtkmm 2 with all dependent packages (libsigc++ 2.x, GTK+ 2.x, cairomm)
4. Sun Microsystems Inc. Java JDK 1.6.0_12+ 32-bit or 64-bit version.
5. X Server must be started and running.

2.3. Mac OS X:

1. Mac OS X 10.5; 32-bit; Apple Java 1.5, 1.6;
2. Mac OS X 10.6; 32-bit or 64-bit; Apple Java 1.6;

Chapter 3. Browser Engines

JxBrowser library allows you to use three browser engines in your Java application: Mozilla Firefox, Internet Explorer and Safari.

As you may see in Chapter 3 the HelloWorld sample uses the following code to create a Browser instance:

```
Browser browser = BrowserFactory.createBrowser();
```

You can use this code when you need to create a default Browser instance for the current platform. The BrowserFactory.createBrowser() method returns the following Browser instance for a specified platform:

- Windows - Internet Explorer
- Linux - Mozilla Firefox
- Mac OS X - Apple Safari

Please refer to the following chapters to find out how to create a specific Browser engine.

3.1. Mozilla Firefox browser

To create Mozilla Firefox browser engine you can use following code:

```
Browser browser = BrowserFactory.createBrowser(BrowserType.Mozilla);
```

This browser type can be used on Microsoft Windows, Linux and Mac OS X (Intel & PPC) platforms.

This browser type doesn't requires Mozilla FireFox application to be installed on operating system.

3.2. Internet Explorer browser

To create Internet Explorer browser engine you can use following code:

```
Browser browser = BrowserFactory.createBrowser(BrowserType.IE);
```

This browser type can be used only on Microsoft Windows platform.

This browser type requires MS Internet Explorer 6.0 or later application to be installed on operating system.

3.3. Safari browser

To create Safari browser engine you can use following code:

```
Browser browser = BrowserFactory.createBrowser(BrowserType.Safari);
```

This browser type can be used only on Mac OS X (Intel) 10.5+ platform.

This browser type requires Safari to be installed on operating system.

3.4. Libraries Dependency

Installation package of JxBrowser contains jar files for all supported platforms (Windows, Mac OS X and Linux) and Browser engines. Below you can find the list of required libraries for each engine for each platform:

Core libraries are required **for all** engines and platforms:

- jxbrowser-2.x.jar*****
- slf4j-api-1.5.8.jar*****
- slf4j-log4j12-1.5.8.jar*****
- log4j-1.2.15.jar*****

Mozilla browser / Windows (only JDK 32-bit):

- engine-gecko.jar
- MozillaGlue.jar
- MozillaInterfaces.jar
- xulrunner-windows.jar
- winpack-3.8.jar*****

Mozilla browser / Linux (JDK 32-bit or 64-bit):

- engine-gecko.jar
- MozillaGlue.jar
- MozillaInterfaces.jar
- xulrunner-linux.jar
- xulrunner-linux64.jar
- tuxpack-0.2.jar*****

Mozilla browser / Mac OS X (only JDK 32-bit or enabled 32-bit mode):

- engine-gecko.jar
- MozillaGlue.jar
- MozillaInterfaces.jar
- xulrunner-mac.jar

Internet Explorer browser / Windows (JDK 32-bit or 64-bit):

- engine-ie.jar
- winpack-3.8.jar*****

Safari browser / Mac OS X (JDK 32-bit or JDK 64-bit):

- engine-webkit.jar

*** Note: library version can be changed in further JxBrowser releases.**

Chapter 4. Navigation to URL

With JxBrowser you can download a web page or HTML document using the following code:

```
Browser browser = BrowserFactory.createBrowser();
browser.navigate("http://www.google.com");
```

The Browser.navigate() method starts document loading, but it doesn't guarantee that the document is completely loaded after the method is executed. To make sure that the document is loaded completely you can use one of the following ways:

1. Use the NavigationListener.navigationFinished() event. This event fires when a navigation operation is finished and a document has been completely loaded and initialized. In pages with no frames or where multiple frames are loaded, this event fires once after loading is complete. Example:

   ```
   browser.addNavigationListener(new NavigationAdapter() {
     @Override
     public void navigationFinished(NavigationFinishedEvent event) {
       // document is loaded completely
     }
   });
   browser.navigate("http://www.google.com");
   ```

2. Use the Browser.waitReady() method. This method blocks current thread execution until the document is loaded completely. Example:

   ```
   browser.navigate("http://www.google.com");
   browser.waitReady();
   ```

   Also please see the Browser.stop(), Browser.refresh(), Browser.goBack() and Browser.goForward() methods.

Chapter 5. Navigation to URL with POST data

During navigation to a specified URL you can also pass POST data. The following sample demonstrates how to do this:

```
browser.navigate("http://www.google.com", "name=Alex&product=Pizza");
```

Chapter 6. Working with Browser Events

Browser engines send notifications about their various events related to navigation, document loading completion, document title or status bar text change, etc.

NavigationListener, ProgressListener, StatusListener and TitleListener provide methods that are called when a browser event occurs.

The following sample demonstrates how to add a navigation started and finished event listener to the browser:

```
Browser browser = BrowserFactory.createBrowser();
browser.addNavigationListener(new NavigationListener() {
   public void navigationStarted(NavigationEvent event) {
   }
   public void navigationFinished(NavigationFinishedEvent event) {
   }
});
```

StatusListener handles browser state change event such as changes of the status text, navigation button state, etc. TitleListener handles document title change events.

Chapter 7. Pop-up windows management

A lot of web-pages use the window.open JavaScript function to open a new pop-up window. JxBrowser provides functionality that allows handle opening a new pop-up window. By default JxBrowser handles opening a new pop-up window and displays it in a separate Swing frame. To change the default behavior you can implement and register your own NewWindowManager. The following sample demonstrates how to do this:

```
Browser browser = BrowserFactory.createBrowser();
browser.getServices().

setNewWindowManager(new NewWindowManager() {
   public NewWindowContainer evaluateWindow
   (
      final NewWindowParams params){
      return new NewWindowContainer() {
         public void insertBrowser(final Browser browser) {
            // creates JFrame in which browser will be embedded
            final JFrame popupFrame = new JFrame();
            Container contentPane = popupFrame.getContentPane();
            contentPane.add(browser.getComponent(), BorderLayout.CENTER);
            // sets window bounds using popup window params
            popupFrame.setBounds(params.getBounds());
            popupFrame.setVisible(true);
            // registers window listener to dispose Browser instance on window close
            popupFrame.addWindowListener(new WindowAdapter() {
               @Override
               public void windowClosing(WindowEvent e) {
                  browser.dispose();
               }
            });
            // registers Browser dispose listener to dispose window when Browser
            // is disposed (e.g. using the window.close JavaScript function)
            browser.addDisposeListener(new DisposeListener() {
               public void browserDisposed(DisposeEvent event) {
                  popupFrame.dispose();
               }
            });
         }
      };
   }
});
```

To suppress all pop-up windows you can use the following code:

```
browser.getServices().setNewWindowManager(new NewWindowManager() {
   public NewWindowContainer evaluateWindow(NewWindowParams params) {
      return null;
   }
});
```

Chapter 8. Working with Browser Dialogs

By default a Browser component displays all dialogs such as JavaScript Alert, Confirmation, Prompt or JavaScript Error. If you want to change this behavior you should take a look at the com.teamdev.jxbrowser.prompt.P interface and the BrowserServices.setPromptService(PromptService promptService) method.

Sometimes you may need to prevent all dialogs from displaying. In this case you just need to set a com.teamdev.jxbrowser.prompt.SilentPromptService implementation as a default PromptService. For example:

```
BrowserServices.getInstance().setPromptService(new SilentPromptService());
```

You can also override a default implementation of PromptService and provide your own behavior for a specified dialog. For example we need to display all dialogs but all Confirmation dialogs should always return CANCEL value:

```
BrowserServices browserServices = BrowserServices.getInstance();
browserServices.setPromptService(new DefaultPromptService() {
   @Override
   public CloseStatus confirmationRequested(DialogParams params) {
      return CloseStatus.CANCEL;
   }
});
```

Chapter 9. JavaScript execution

With Browser.executeScript() method you can execute any JavaScript code on a loaded web page. For example:

```
browser.executeScript("document.write('Hello from JxBrowser!');");
```

This method also returns the result of JavaScript code execution as a String value. So you can execute JavaScript code like this:

```
browser.navigate("http://www.google.com");
browser.waitReady();
String title = browser.executeScript("document.title");
System.out.println("title = " + title);
```

Chapter 10. Cookie management

Each browser engine has its own cookie storage. You can get access to the cookie storage of a specified browser engine you can use one of the following ways:

```
Browser browser = BrowserFactory.createBrowser(BrowserType.Mozilla);

// Get access to the Mozilla engine cookie storage
HttpCookieStorage mozillaCookieStorage = browser.getCookieStorage();

// Get the list of all available cookies
List<HttpCookie> cookies = mozillaCookieStorage.getCookies();
```

or

```
BrowserServices browserServices = BrowserServices.getInstance();

// Get access to the Mozilla engine cookie storage
HttpCookieStorage mozillaCookieStorage = browserServices.getCookieStorage(BrowserType.Mozilla);

// ... working with cookie storage
```

Chapter 11. Change user-agent string

The user-agent string can be changed programatically via Browser.setUserAgent method. For example:

```
Browser browser = BrowserFactory.createBrowser(BrowserType.Mozilla);

// Set custom user-agent string for Mozilla browser instance
browser.setUserAgent("My UserAgent String");
```

To restore a default user-agent string please use the following way:

```
Browser browser = BrowserFactory.createBrowser(BrowserType.Mozilla);

// Restoring a default user-agent string for Mozilla browser instance
browser.setUserAgent(null);
```

Chapter 12. Using JxBrowser with Java Web Start

This section describes how you can deploy your application that uses JxBrowser with Java Web Start technology.

Below you can find what JxBrowserDemo JNLP (JxBrowserDemo.jnlp) file looks like:

```
<jnlp spec="1.5+" codebase="http://enclave:8080" href="JxBrowserDemo.jnlp">
  <information>
    <title>JxBrowserDemo</title>
    <vendor>TeamDev Ltd.</vendor>
    <homepage href="http://www.teamdev.com/jxbrowser"/>
    <description>JxBrowser Demo Application</description>
    <offline-allowed/>
  </information>

  <security>
    <all-permissions />
  </security>

  <application-desc main-class="com.teamdev.jxbrowser.demo.JxBrowserDemo" />

  <resources>
    <jar href="lib/jxbrowserdemo.jar" main="true" />

    <jar href="lib/jxbrowser-2.x.jar * " />
    <jar href="lib/engine-gecko.jar" />
    <jar href="lib/MozillaGlue.jar" />
    <jar href="lib/MozillaInterfaces.jar" />
    <jar href="lib/log4j-1.2.15.jar * " />
    <jar href="lib/slf4j-api-1.5.8.jar * " />
    <jar href="lib/slf4j-log4j12-1.5.8.jar * " />
  </resources>
  <resources os="Windows">
    <j2se version="1.5+" initial-heap-size="128m" max-heap-size="512m"/>
    <jar href="lib/engine-ie.jar" />
    <jar href="lib/winpack-3.8.jar * " />
    <jar href="lib/xulrunner-windows.jar" />
  </resources>
  <resources os="Mac OS X">
    <j2se version="1.5+" initial-heap-size="128m" max-heap-size="512m" java-vm-args="-d32"/>
    <jar href="lib/engine-webkit.jar" />
    <jar href="lib/xulrunner-mac.jar" />
  </resources>
  <resources os="Linux">
    <j2se version="1.6+" initial-heap-size="128m" max-heap-size="512m"/>
    <jar href="lib/tuxpack-0.2.jar * " />
  </resources>
 <resources os="Linux" arch="ppc i386 i686">
    <jar href="lib/xulrunner-linux.jar" />
  </resources>
 <resources os="Linux" arch="x86_64 amd64">
    <jar href="lib/xulrunner-linux64.jar" />
  </resources>
</jnlp>
```

*** Library version can be changed in further JxBrowser releases.**

> **Important:** Please note that JNLP file must contain the following options to successfully start JxBrowser library:

```
<security>
  <all-permissions/>
</security>
```

JxBrowser Online Demo JNLP file is configured to use the following libraries:

- jxbrowserdemo.jar - the JxBrowser demo application library with the main class;
- log4j-1.2.15.jar, slf4j-api-1.5.8.jar, slf4j-log4j12-1.5.8.jar - JxBrowser logging libraries;
- jxbrowser-2.x.jar - JxBrowser API library;
- engine-ie.jar - Microsoft Internet Explorer engine support;
- engine-gecko.jar - Mozilla Firefox engine support;
- engine-webkit.jar - Apple Safari (WebKit) engine support;
- winpack-3.8.jar - provides access to Microsoft Windows native platform API;
- tuxpack-0.2.jar - provides access to Linux native platform API;
- xulrunner-windows.jar, xulrunner-linux.jar, xulrunner-linux64.jar, xulrunner-mac.jar - Mozilla XULRunner packages for Microsoft Windows, Linux and Mac OS X;
- MozillaGlue.jar, MozillaInterfaces.jar - third-party Java libraries required for Mozilla Firefox engine;

To deploy your application via Java Web Start, please use the following steps:

1. Pack your application classes into application JAR file(s) (demo.jar);
2. Put the JxBrowser runtime license file (teamdev.licenses) into META-INF folder of your application JAR file. For this you can use the following ANT script:

   ```
   <jar destfile="${demoJarName}">
     <fileset dir="${classes}" includes="**/**" />
     <zipfileset dir="${srcPath}/lic" includes="teamdev.licenses" prefix="META-INF" />
     ...
   </jar>
   ```

   where:

   ${classes} - path to class files of demo application;

   ${srcPath}/lic - path to licence file teamdev.licenses;

   Or you can use the jar tool to update your application JAR file (demo.jar). For this you should create META_INF folder and copy license file (teamdev.licenses) to META_INF folder. After that you should run following command from command line:

   ```
   jar uf demo.jar META-INF\teamdev.licenses
   ```

3. Sign all your application JAR files. Please note that by default all JxBrowser libraries are signed with TeamDev signature. For this purpose, you can use the following ANT templates:

   ```
   <project name="Sample" default="build" basedir=".">
     <target name="demojar">
       <property name="demoJarName" value="demo.jar"/>
       <signjar jar="${demoJarName}" alias="your_alias" keystore="your_keystore"
           keypass="your_keypass" storepass="your_storepass"/>
     </target>
   </project>
   ```

where:

your_alias, your_keystore, your_keypass, your_storepass - signjar parameters. You can see their description in Apache Ant User Manual at http://ant.apache.org

Or you can use the jarsigner tool to sign your application JAR file (demo.jar):

```
jarsigner -keystore your_keystore_file -storepass your_storepass -keypass your_keypass demo.jar your_alias
```

4. Create and configure your application JNLP file;
5. Upload your application JAR files, JxBrowser JAR files and JNLP file to a Web-server.

Please note that JxBrowser libraries are already signed with TeamDev signature. So, in order to use already signed JxBrowser libraries with your application JNLP file you can use the <extension> tag. So the result JNLP file should contain only your application libraries and another JNLP file (included through extension tag) which contains only JxBrowser libraries.

This technique is demonstrated in the example below. The first file is an application JNLP file (demo.jnlp):

```
<jnlp spec="1.5+" codebase="http://codebase" href="demo.jnlp">
<information>
   <title>Demo Application</title>
   <vendor>TeamDev Ltd.</vendor>
   <description>Demo Application</description>
   <offline-allowed/>
</information>
<security>
   <all-permissions/>
</security>
<resources>
   <j2se version="1.5+" initial-heap-size="64m" max-heap-size="256m"/>
   <!-- demo.jar is your jar file signed with your own signature-->
   <jar href="demo.jar"/>
   <extension name="jxbrowser" href="JxBrowser.jnlp"/>
</resources>
<application-desc main-class="main_class"/>
</jnlp>
```

The <extension> tag above makes the reference to the other jxbrowser.jnlp file which has the following structure:

```
<jnlp spec="1.5+" codebase="http://codebase" href="JxBrowser.jnlp">
<information>
   <title>JxBrowser resources</title>
   <vendor>TeamDev Ltd.</vendor>
   <homepage href="http://www.teamdev.com/jxbrowser"/>
   <offline-allowed/>
</information>
<security>
   <all-permissions/>
</security>
<component-desc/>
<resources>
      <jar href="lib2/jxbrowser-2.x.jar" />
      <jar href="lib2/engine-gecko.jar" />
      <jar href="lib2/MozillaGlue.jar" />
      <jar href="lib2/MozillaInterfaces.jar" />
      <jar href="lib2/log4j-1.2.15.jar" />
      <jar href="lib2/slf4j-api-1.5.8.jar" />
      <jar href="lib2/slf4j-log4j12-1.5.8.jar" />
   </resources>
   <resources os="Windows">
      <j2se version="1.5+" initial-heap-size="128m" max-heap-size="512m"/>
      <jar href="lib2/engine-ie.jar" />
```

```
      <jar href="lib2/winpack-3.8.jar" />
      <jar href="lib2/xulrunner-windows.jar" />
   </resources>
   <resources os="Mac OS X">
      <j2se version="1.5+" initial-heap-size="128m" max-heap-size="512m" java-vm-args="-d32"/>
      <jar href="lib2/engine-webkit.jar" />
      <jar href="lib2/xulrunner-mac.jar" />
   </resources>
   <resources os="Linux">
      <j2se version="1.6+" initial-heap-size="128m" max-heap-size="512m"/>
      <jar href="lib/tuxpack-0.2.jar *" />
   </resources>
 <resources os="Linux" arch="ppc i386 i686">
      <jar href="lib/xulrunner-linux.jar" />
   </resources>
 <resources os="Linux" arch="x86_64 amd64">
      <jar href="lib/xulrunner-linux64.jar" />
   </resources>
</jnlp>
```

Chapter 13. Using JxBrowser with Java Applet

You can use JxBrowser in your Java Applet application in the same way as a standard Java library.

JxBrowser requires access to the file system, so you should sign all JARs in your Java Applet application including all JxBrowser library JARs.

By default all JxBrowser JARs are signed with TeamDev signature. So, you need to remove TeamDev signature before signing with your one. To remove TeamDev signatures you should remove the META-INF\TEAMDEVC.RSA and META-INF\TEAMDEVC.SF files from each JAR from JxBrowser distribution package.

When all your application JARs are signed including all required JxBrowser JARs you can configure your APPLET tag. For example:

```
<APPLET code="Application.class" codebase="." archive="application.jar,<List of required JxBrowser JARs>"
   WIDTH=800 HEIGHT=400>Browser doesn't support Java Applets.</APPLET>
```

Chapter 14. JxBrowser System Properties

JxBrowser library uses several System Properties which can be useful for you. You should set up these properties before creating a browser instance in the following way:

```
System.setProperty("property_name", "value");
```

Basic properties

teamdev.license.info [true, false] - allows enabling or disabling the license information printing in console.

IE specific properties

JExplorer.runInIsolatedProcess [true, false] - allows enabling or disabling InIsolatedProcess mode for IE engine. In this mode each IE engine instance will be running in a separate native process - JExplorer32.2.2.10.exe. On JDK 32-bit this mode by default is enabled. To disable this mode you can set this property to false.

Mozilla specific properties

jxbrowser.xulrunner.dir [absolute or relative path] - using this property you can configure the path to the directory where XULRunner will be extracted. By default XULRunner will be extracted at first start into the USER_HOME\.JxBrowser directory.

jxbrowser.xulrunner.profile.dir [absolute or relative path] - using this property you can configure the path to the profile directory. By default profile directory will be created at first start into the USER_HOME\.JxBrowser directory.

jxbrowser.xulrunner.use.profile [true, false] - allows creating engine with/without profile directory. By default profile directory will be created.

GRE_HOME [absolute or relative path] - allows specifying the path to the directory where XULRunner is located. By default XULRunner will be extracted at first start into the USER_HOME\.JxBrowser directory. XULRunner extraction may take several seconds. To avoid extraction you can provide the path to the directory with already extracted XULRunner.

jxbrowser.plugin.dir [absolute or relative path] - allows specifying the path to the directory where Mozilla plugins are located.

Chapter 15. Support

JxBrowser Support Site

You can find more information about how to use JxBrowser library at JxBrowser Support Site

JxBrowser Samples

You can find JxBrowser samples on the following resource: JxBrowser Samples

Reporting Problems

If you have any questions regarding JxBrowser, please e-mail us at: jxbrowser-support@teamdev.com

JxBrowser Forum

If you would like to discuss any topics related to JxBrowser, please visit our support forum at: JxBrowser forum